AFRL-IF-RS-TR-2006-134
Final Technical Report
April 2006

MITIGATING THE INSIDER THREAT USING HIGH-DIMENSIONAL SEARCH AND MODELING

Telcordia Technologies, Inc.

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. S470

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2006-134 has been reviewed and is approved for publication

APPROVED: /s/
DAVID KRZYSIAK
Project Engineer

FOR THE DIRECTOR: /s/
WARREN H. DEBANY, Jr.
Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 2006	Final Jul 05 – Jan 06

4. TITLE AND SUBTITLE

MITIGATING THE INSIDER THREAT USING HIGH-DIMENSIONAL SEARCH AND MODELING

5. FUNDING NUMBERS

C - FA8750-04-C-0249
PE - 62301E/62304E
PR - S470
TA - SR
WU - SP

6. AUTHOR(S)

Eric VanDenBerg, Shambhu Uphadyaya, Phi Hung Ngo, Muthu Muthukrishnan and Rajago Palan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

PRIME:
Telecordia Technologies, Inc.
Piscataway NJ

SUB:
SUNY Buffalo
Buffalo NY

SUB:
Rutgers University
Davidson Road
Piscataway NJ

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFGA
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-2006-134

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: David Krzysiak, IFGA, 315-330-7454, David.Krzysiak@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; Distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*

In this project a system was built aimed at mitigating insider attacks centered around a high-dimensional search engine for correlating the large number of monitoring streams necessary for detecting insider attacks. Further accomplishments in this project include an insider attack modeling and analysis tool called MAPIT, developed by SUNY Buffalo, and a novel sketch-based anomaly detection sensor developed by Rutgers University, which can be used for detecting anomalies in IP source/destination addresses, as well as for defining small-space user profiles, e.g., file accesses.

14. SUBJECT TERMS

Insider threat, high-dimensional search, reasoning

15. NUMBER OF PAGES

52

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

Table of Figures

1 Summary

In this project, “Mitigating the Insider Threat via High-dimensional Search and Modeling”, we have built a system aimed at mitigating insider attacks consisting of the following components: a High-dimensional search engine, a graphical Insider Modeler and Analyzer, a Response Engine, and a Sensor Network.

Insider attacks are significantly harder to defend against than outsider attacks. Part of the problem is that the insider has authorized access to the system, and often knowledge of target resources. Consequently, fusion or correlation of possibly hundreds of different alerts is required. This correlation is accomplished using high-dimensional search and anomaly detection. Underlying assumptions are that the insider will trigger at least some sensors prior to and during his attack. This assumption best fits an insider with up to moderate knowledge about his malicious goal. The system is quite effective in detecting attacks committed by such insiders, as was shown in a demonstration at the July 2005 PI meeting.

In testing the system, we have mainly focused on information ex-filtration attacks. In such attacks, we gave the ‘insiders’ information about the nature of the malicious target, a file with sensitive (military) information, and approximate location information. With this kind of information, we detected 3 out of 4 insider attacks on our test-bed in the July demonstration, while the 4th insider did not get near a malicious goal.

We have integrated a response engine in our system to generate an automatic response to detected attacks. While originally envisioned to thwart attacks in a time span of minutes, the detection/response cycle is in fact fast enough to generate an automatic response in seconds, as was shown in the red team exercise on Nov. 4. This exercise mostly focused on the ability of insiders to evade sensors, and to accomplish reading or altering of the information before the response had been activated. Some one-shot attacks were possible, and the system was not yet fast enough to thwart those. We also were able to take some of the lessons learned from this exercise and improve sensors, as well as improve interaction between detection and response engine.

SUNY Buffalo has developed a graphical insider attack modeling and analysis tool, called MAPIT. This tool has been used e.g. to heuristically find the most likely attack paths an insider can take to reach his/her malicious goal(s). It has been tested on the Telcordia testbed, as well as on (part of) the Hackfest 2004 network.

A further innovation of note accomplished in this project is a novel sketch-based anomaly detection sensor developed by Rutgers University, which can be used for detecting anomalies in ip source / destination addresses, as well for defining small-space user profiles for e.g. file accesses.

2 Introduction

Insiders pose a substantial threat by virtue of their knowledge of and access to their employers' systems and/or databases, and their ability to bypass existing physical and electronic security measures through legitimate means [1]. The insider threat has been recognized as a critical security problem, although the statistics reported are varying (from 70% to 59% of successful intrusions are committed by insiders, see e.g. [12, 14]). Here, an insider is defined in as anyone who is or has been authorized access to an information system.

Several workshops have been devoted to the subject; see e.g. [10], but until recently, real world case studies and data were scarce. During the course of this project, two studies by the US Secret Service/CERT [1,11] and a survey [12] have appeared. The studies target illicit cyber activity in the banking and finance industry, and computer system sabotage in critical infrastructure sectors respectively. Some of the notable observations from the banking and finance study are, that insiders planned their actions, the actions often did not require much technical sophistication, and the attacks were detected by various means, including system logs to obtain attacker identities in 74% of the cases.

Detecting insider malfeasance requires fusion of possibly hundreds of different measurements and alerts to get the overall picture of the system. Current approaches to anomaly detection cannot handle hundreds of different attributes simultaneously. Furthermore, insiders may be in a position to subvert or fool some sensors installed in the system, requiring us to work with partially falsified or missing sensor data. Insiders may be aware of pre-defined attack models, requiring us to go beyond the simple signature-based approach that is being used to detect them. Even when we do suspect an insider attack, we may not be able to respond for fear of "collateral damage" which may cause more harm by itself. Finally, we do not have systems that can detect patterns of malfeasance that can scale to Internet-sized networks. We propose a new solution methodology that addresses the above.

Our proposed approach to insider threats is inspired by the human cognitive model of recalling complex past experiences similar to a given situation and synthesizing a response that takes advantage of that experience. The approach has focused on the following research goals:

- Developing a sensor architecture that can monitor a very large set of different system parameters from a very large system that range from physical to network to application layer, including end-host sensors, making it much harder for insiders to act without triggering at least some sensor alerts.
- Reducing the dimensionality problem created by the very large amount of sensor data using a high-dimensional Search Engine (that is traditionally used in searching documents on the web) to create a ranked list of annotated states from the network's history that are sufficiently similar to the current state.
- Detecting insider threats that are similar to threats observed in the past based on the insight that most attacks tend to be similar or variations of past attacks.

- Learning new states over time by maintaining a network history repository that contains historical states that are annotated with attack and precursor information.
- Developing an Insider threat modeler and analyzer tool, which uses a novel graph-theoretical approach to model and analyze malicious insiders in an organization network as the first layer of defense.
- Responding to threats that have been detected within minutes and thwarting those attacks to protect critical services: a Response Engine that performs an impact analysis of the potential attack on critical services and synthesizes a response that minimizes collateral damage. The response engine concepts were originally developed through funding from DARPA's Dynamic Coalitions program.

This innovative approach is based on several underlying assumptions:

- Since the detection/response cycle is envisioned to be in the order of minutes, this approach is most suitable for detecting slow, stealthy insider attacks, rather than 'instant' attacks such as worm-attacks. However, often even fast, noisy insider attacks may have detectable precursors long in advance [1]. For instance, it has been noted that insiders often plan their actions long in advance, and may even discuss them with co-workers or family members.
- Since our approach relies in part on the availability of alerts from a large number of sensors, another assumption is that the inside attacker will always trigger at least some sensor alerts, and cannot subvert or circumvent them all. This may rely on appropriate policies being in place.

2.1 System overview

We divide the entire process of combating insider threat into three stages:

a) Pre-Attack Stage
   In the pre-attack phase, we represent organizational information in a graph-based model that allows us to analyze statically any pre-existing insider threats. The insider threat scenarios so discovered can be used to harden policies, place sensors, and train our detection system by annotating simulated attack states.
b) Attack Detection Stage
   Using a search engine, we create annotated clusters of similar states in a high-dimensional space, which in turn, enables us to categorize the current state of the system in terms of attack precursors. For this purpose, we represent and store the historical states of the system with annotations of attacks, precursor functions, and response strategies.
c) Post-Detection Stage
   We then thwart these attacks by doing an impact analysis of the potential attacks on critical services and by synthesizing and implementing appropriate network reconfigurations that minimize collateral damage.

Our System Architecture is shown in Figure 1 below:

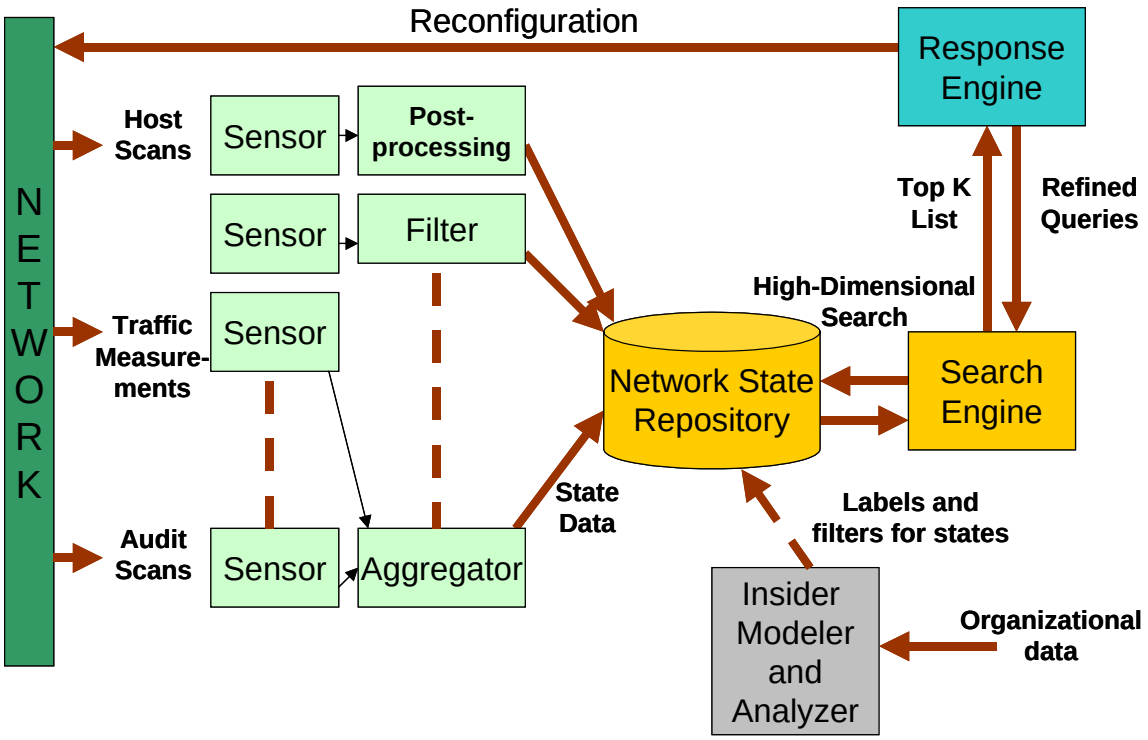

Figure 1: System Architecture

The architecture shown in Figure 1 incorporates a large number of **Sensors** that monitor a large number of system parameters and can provide alerts when suspicious behavior is observed. The architecture is open so that a wide variety of sensors can be incorporated. The goal is to make sure that such a large number of parameters are monitored so that no insider threat can go unnoticed without generating at least some type of an alarm. Aggregators, filters, and detectors are used for reducing the problem state space without losing key information.

The **Network State Repository** in Figure 1 contains historical states of the system that are annotated with attack, precursor and response information. The core of the network state is a set of sensor alerts. This data gets updated when new previously unknown attacks are observed so that the system can learn over time.

The **Search Engine** in Figure 1 is based on dimension-reduction techniques such as Singular Value Decomposition (SVD). One of the first large-scale information retrieval engines, LSI (based on SVD) was developed at Telcordia Technologies (then Bellcore). This gave us confidence that SVD-based methods will be successful in the context of high-dimensional anomaly and attack detection. Similar to the searched document list, the search engine will give us the list of states that are most similar to the query state. Our method does not depend on pre-scripted attack models and at the same time is able to classify the current state of the network in terms of attack precursors.

The **Insider Modeler and Analyzer** component serves multiple purposes in the overall scheme. A security analyst can develop a layout of the organization's computation and personal resources and provide this as an input to the component, which in turn, statically produces scenarios or weaknesses leading to successful insider attacks. Consequently, the infrastructure can be hardened against these attacks. This is its main functionality. In order to enhance attack detection, these scenarios can also be used to sensitize the network repository and guide the placement of sensors.

The **Response Engine** in Figure 1 applies to the problem of insider threat the concept of the logic-based policy engine that was developed in the Telcordia's "Smart Firewalls" project funded by DARPA's Dynamic Coalitions program. Given the top K list from the search engine listing the most likely attacks, the response engine logically analyzes the impact of these attacks on critical services. The impact analysis enables us to infer the overall attack risk. The response engine generates the necessary reconfigurations in the network to thwart the attack, which can be pushed into the network automatically, thus protecting critical services before they are attacked. The response engine can respond to events that are low certainty but potential high impact. In the particular case where we cannot classify with high certainty that the state is an attack precursor, but the impact on critical services is high, the response engine can act proactively and reconfigure the network with minimal damage to legitimate services.

The following section discusses the design and implementation of each of the components of our approach in more detail.

3 Methods, assumptions and procedures

3.1 Sensor Network

To defend any system we have to be able to monitor it well and at all times. For this project, we have install sensors at multiple system layers, to monitor applications, servers, hosts and other devices on which the functioning of the service depends. The data and alerts generated by the sensors comprise a high-dimensional state space that our developed search engine (discussed in the next section) can search. The installed sensors include both end-host sensors (applications, audit logs, etc) and network sensors (router traffic, aggregate and flow-data).

3.1.1 Architecture

Our primary goals in designing the sensor network for this project were scalability and extensibility, e.g. the ability to add arbitrary sensors. Furthermore, to enable forensic analysis / refined search, the raw alerts from the sensors should be available to the back-end search process. We prefer to concentrate the 'intelligence' of the system in the search engine, and not burden the sensors beyond collecting data. However, there are occasions when it is useful to filter or aggregate data for a noisy sensor or set of sensors, in order to

sanitize or reduce the volume of data. This has led us to the basic sensor architecture shown in
Figure 2.

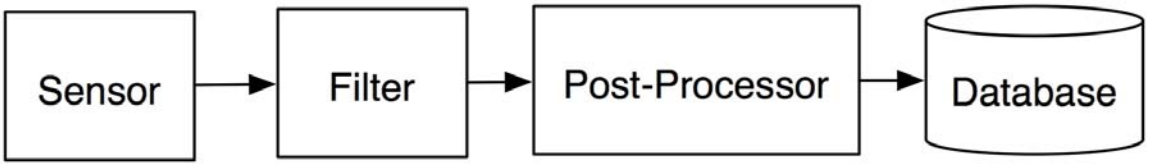

Figure 2: The Basic Sensor Network

To meet the goal of *scalability*, we first perform data volume reduction by letting the sensors do alert-driven reporting. Alerts can be triggered either e.g. by a signature or by exceedance of the sensor values of a given threshold for 'normal behavior'. Furthermore, they can report a new event, such as a user login or logout. This alert-driven reporting is meant to reduce data volume without significant information loss. Furthermore, we isolate the sensors and their post-processing as much as possible. Thus, a typical sensor will perform only minimal post-processing, which consists of any filtering specific to that filter followed by normalization of the alerts to common format. The filtering is generally simple in nature: e.g. to collect together identical or near identical alerts and produce a single alert with a count. In cases where no filtering is required or where no post-processing is required, those steps would be removed from the chain.

The stream of alerts is joined when it enters the database. We chose this mode of operation because database technology is readily available for handling high volumes of transactions concurrently. Additionally, this approach serves to decouple the sensor network from the search system, so the sensors and search engine can operate in parallel. For the data collection front-end of our system, the sensor, filter, and post-processor boxes shown in
Figure 2 will be repeated several times with a common database. Furthermore, a cornerstone of our design is to be able to incorporate data from as many different sensors as possible, some of which may pre-aggregate as a natural function, as depicted in Figure 3

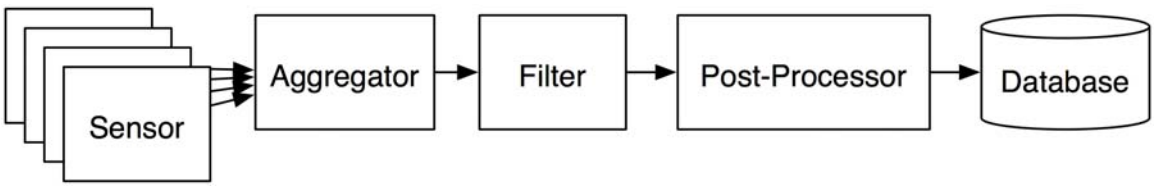

Figure 3: An aggregated sensor

We have re-used available sensors as much as possible. The decision to work with alert-driven sensor reporting also aids the goal of *extensibility*, since our architecture is designed to easily integrate several existing alert-driven sensors. Since the Intrusion Detection Message Exchange Format (IDMEF) is rapidly becoming accepted as the 'lingua-franca' of IDS message formats, we will adopt this format for alert reporting of individual sensors. Any sensors that do not natively produce messages according to this standard can be adapted. Officially, "The purpose of the Intrusion Detection Message Exchange Format (IDMEF) is to define data formats and exchange procedures for sharing information of interest to intrusion detection and response systems, and to the management systems which may need to interact with them"[3].

We have also used IDMEF as the format for reporting attacks detected by the Search Engine to the Response Engine.

The alerts are stored in the database in an IDMEF-based schema. By using a standard format, we are also able to take advantage of existing tools for handling alerts. This allows our database to be the central database for security monitoring rather than requiring that alerts be duplicated to another database for other processing. Additionally, as more and more tools produce IDMEF as their native format, we are able to delete the normalization step in our pipeline to potentially reduce processing requirements and latency.

As a matter of fact, different sensors already available and producing IDMEF output may interpret certain fields in the IDMEF XML standard slightly differently. However, as we will discuss in the design of the High-dimensional network state description and Search Engine below, the effect of different interpretations of the IDMEF standard on the search engine results will be kept to a minimum. This is because it is the 'pattern of alerts' that guides the search, rather than the actual content of the IDMEF. Important fields of the IDMEF message, which do affect search (e.g. "Source", "Classification" and "Target"), are unlikely to be open to interpretation.

To implement large parts of this design, we have leveraged the approach of the Prelude Project. Prelude (www.prelude-ids.org) is an IDS management system available as open source. It has a structure similar to ours for managing sensors. A key functionality of Prelude is collecting alerts from a variety of sensors and inserting them into a database. Our evaluation showed no concurrency issues and using an existing tool allowed us to focus more of our attention on more innovative part of our system.

Prelude can deal with sensors in one of two ways. A sensor (or a post-processor; in fact, also our Search Engine) can be made aware of Prelude. This involves adding some code that will connect to the Prelude manager (the central process that collects alerts) and calling routines to marshal and transmit alerts. The second approach is to use *prelude-lml* or the Log Monitoring Lackey. This is a process that watches a log file for changes. It matches those changes against a set of user provided regular expressions and ``accepts'' lines that match. Based on constant data (such as the type of alert typically) and

information from the regular expression (e.g. an IP address, user ID or log-in name), it builds an alert and sends it to Prelude.

Externally, Prelude uses IDMEF as its alert format. Because the full XML for alerts is inefficient to transmit and relatively expensive to parse, it uses a binary implementation of IDMEF internally to speed transmission and processing. This approach worked well for as we could use available IDMEF tools as needed, but did not have to pay all of the cost of carrying that format around.

3.1.2 Sensor Categories used for Insider Attacks

3.1.2.1 Network-based IDS

Network-based IDS (NIDS) have the desirable capability of sensing attacks on multiple targets using a single sensor. Both signature-based NIDS such as Snort (www.snort.org, for which the Prelude team has provided a patch to report in Prelude's binary IDMEF format) and anomaly based NIDSs based on e.g. packet headers) are of use. Signatures can for instance be used to detect known attacks/vulnerabilities, or, particularly for insiders, e.g. honeytoken [2] exploits. The definition of a honeypot is 'an information system resource whose value lies in unauthorized or illicit use of that resource'. A honeypot is typically a computer, whereas a honeytoken can be any digital entity, such as a spreadsheet, word processor document, or login and password.

To help detect previously unknown attacks, Rutgers University has developed novel 'sketch' based network anomaly detectors based on any attribute ϕ from an IP packet header, e.g. one from (time, sourceIP, destinationIP, size, srcport, destport). For example, let the attribute ϕ which is monitored be the Src IP address. Then the signal S has domain $[0,...,|U|-1]$, where $|U|=2^{32}$ and $S[i]$ is a given measure at i, such as the number of connections, packet counts, bytes transferred, etc. to Src IP address i during a given time interval. At the high level, we consider two signals, S_0 (normal) and S_1 (current). The problem is to detect when a) S_0 and S_1 .differ significantly, and b) find i such that $S_1[i]$ and $S_0[i]$ differ significantly, in this case, i is the 'culprit' or target. The Rutgers team has focused on version b) of the problem. Significant change could mean large absolute or relative difference in case of aggregates such as counts or the Jacard coefficient [18] between sets.

Keeping exact measure of S for each $i \in U$ requires large space overhead for dense signals: $O(|U|)$. However, in most applications, it suffices to produce approximate answers to detect big changes in the signal. An innovative solution is found using a streaming algorithm. By 'streaming algorithm' [19], we mean a method that works in sub-linear space, and handles each update efficiently, typically in sub-linear time. Although such algorithms cannot solve most problems on input signal S exactly, they will succeed with probability at least $1-\delta$ at being accurate to some prescribed error ε for some small parameters δ and ε. Note: for the insider threat, often the number of IP addresses (e.g. in a military/enterprise network) may be sufficiently small to implement

the full space, exact signal. Furthermore, the change detection can be periodic or instantaneous. If a small delay can be tolerated, experiments on the MIT Lincoln Labs IDS evaluation data for 1999 show that the periodic, sketch based solution has great space/speed advantages, and has almost as accurate detection as the full space solution. This result creates very good confidence in the potential for these sketch / group testing based methods.

3.1.2.2 Host-based IDS

In a similar vein, we will make use of host-based IDS (HIDS) technology. One of the difficulties in catching an insider is that he/she often knows the machine that he wishes to attack and may even have access to that machine or be able to gain access to the machine via social engineering rather than launching an attack over the network. Thus, while it is critical to monitor the network for less sophisticated insiders, the more dangerous insider may not leave any "footprints" within the network. Moreover, if network traffic is encrypted, signature-based network-based IDS are not effective. However, we expect that insiders who are making illegitimate accesses to data will likely stray from the normal behavior patterns of the account they are using at some point. This might include searching the file systems as opposed to quickly moving to the normal work environment or otherwise trying things to see what will happen as opposed to behaving like someone who has been trained on a system.

We have made use of HIDS-tools such as the file-integrity checker samhain (http://la-samhna.de/samhain/)), which already include the semantics of specific anomalous behaviors (e.g. those exhibited by root-kit attacks). Samhain can report alerts to a Prelude manager in its binary IDMEF format. It periodically checks files to see if various attributes such as checksum or access times have changed. Modifications are compared with a user specified policy that can be as detailed as per-file. If a modification falls outside of the policy, an alert is generated. While samhain can alert that a modification has occurred (e.g. on a read access to a file), it does not include information by whom.

3.1.2.3 Audit / Application log sensors

We have several audit- and application log sensors for Linux hosts, and build user-profiles for anomaly detection, in order to detect 'straying from normal behavior'. These have been implemented using Prelude's flexible 'Log Monitoring Lackey' prelude-lml, a sensor which continuously scans a specified list of log-files, compares entries against a given list of regular expressions, and sends an alert to Prelude manager (and the IDMEF based database) upon finding a match.

In an attempt to avoid the latency of samhain and to more accurately track users, we concluded that we needed support from the kernel. Although we wrote LSM code as a baseline sensor early on in the project, for the Red Team Exercise we re-tasked SELinux as a demonstration of what could be done. In practice, keeping an SELinux security policy up-to-date on a working machine is a difficult task. However, in permissive mode SE-Linux will issue a warning message describing what policy-based actions it would have taken if it had been in enforcement mode. Moreover, those messages contain information including what class of user is attempting to perform those operations. In

order to make use of this, we modified the standard security policy supplied with Fedora to have a class per user so that we could track-back an access to a particular user. Additionally, we added files of interest to the security policy so that on particular types of access, we could generate an alert. We added some regular expressions to prelude-lml to take the log messages produced by SELinux and convert those into system alerts.

We have modified a version of PAM (the Pluggable Authentication Modules system used for a variety of authentication tasks under Fedora Linux) to generate e.g. an alert on presentation of a honeytoken during a login attempt. This sensor was not a particularly realistic sensor for our system (typically a honeytoken login attempt does not need to be correlated to be considered an attack), but helped to demonstrate the basic framework during the first PI meeting in January 2005. Additionally, we trained the system so that Snort would alert on seeing the honeytoken cross the network (e.g., in an email planting the honeytoken), but the search engine would ignore the alert since it was not correlated with a PAM alert. When a login attempt was made across the network (via some clear channel such as telnet), the combined Snort and PAM alerts would cause a correlated alert to be raised. As noted above, prelude-lml is used to monitor log files. For example, later in the project, as we started to try to determine a responsible user to allow a response, we wanted to learn who was logged into a machine. We used prelude-lml to monitor the log files produced by PAM and generate alerts on login and logout events.

3.1.3 Command-line monitoring

To create a profile of normal user behavior, a basic assumption/observation was that a typical user has a vocabulary of commands with which they are familiar and typically use commands from that vocabulary. Moreover, they have typical sequences (such as gcc, myprog, gdb, vi or cd, ls, cd, ls). Our examples are command line oriented in keeping with our implementation. We believe that one could instrument the operating system and possibly window system to generate a similar sequence of events on program launches to get a similar effect. The underlying assumption was that an attacker is likely to have a different command-vocabulary or command-sequences and may have to use some unusual commands, for example, to perform privilege escalation. Command line activity has previously been used to detect so called ‘masqueraders’, people using others’ computer accounts. However, most of the work has concentrated on evaluating large blocks of commands, and / or ‘enriched’ command-lines, which need much training data. In this project, in order to thwart insider attacks such as information ex-filtration, we need an anomaly detector which can report unusual behavior in real-time, without delay.

In order to monitor ‘unusual commands’, we have adapted a n-gram based anomaly detector, Stide [17]. Stide is a tool that was originally used to signal abnormal system call sequences. At its core, though, it takes a series of lines each with two columns. The first column is a stream identifier. Stide can process data from multiple streams and will consider each stream independently for purposes of learning or evaluating sequences. The second column is an integer representing one event in the sequence.

In order to implement the real-time command line monitoring system, we made some minor modifications to bash, the ``standard'' shell available with Linux. Our modified version of bash writes its process ID and the inode of the command run each time the user ran a command. In the case of built-in commands, we had a map from the built-ins to small integers. Mapping commands to integers this way had the advantage of being very simple and very quick to implement.

Finally, we had a configuration file to determine for each user whether they should be in training mode or in detection mode. In training mode, we merely collected the records produced by bash for later use. When sufficient training had been collected, we ran stide's training mode to allow it to collect sequences. Those sequences were then used in detection mode with our modified bash sending commands as they were typed to stide. When an unusual sequence was seen, our version of stide had been modified to issue a Prelude alert.

Combining samhain and stide gave us a good set of alerts to correlate during our first demonstration at the July 2005 PI meeting. As noted, stide would sometimes produce alerts because a user used an unusual command. samhain would sometimes produce alerts when a user made a legitimate access to a file. However, when we asked some people who were not on the project to try to find and gain access to some ``sensitive'' files, they tended to use unusual commands and access guarded parts of the system allowing us to provide a correlated alert corresponding to the attack.

After the Red Team Exercise, late in the project, we have developed a sequential Naïve Bayes detector as an alternative anomaly detection method. Naïve Bayes was the best performing anomaly detector when evaluated on command-line blocks of 100 commands in [16]. The innovation here is the use of a sequential test, in order to have real-time detection. We have used the same modified bash version to collect command line information, and only switched the detection module.

Furthermore, we have noticed that the Rutgers sketches can also be used to create small space profiles for host-based user behavior, such as file access profiles. This is another use of this technique to provide insider attack sensors.

3.2 Search Engine

The Search Engine's function is to take the current network state (described as a collection of sensor alerts) and find the most relevant similar historical network states (also described as a collection of sensor alerts), which are annotated with attack classification and response information, and do this in near-real time. The current 'network state' is given by the set of alerts which have arrived in the last sliding time interval of x seconds (x=20 seconds, 60 seconds for example) The network state is updated every few (=2) seconds in our implementation, but this frequency can be somewhat increased without overloading the engine.

The first step in searching or anomaly detection is to translate the current alert set to a vector. In this vector, each possible occurring alert by each sensor type is assigned a coordinate. To populate the vector network state description, there are several options:

a) We add 1 to the vector coordinate corresponding to an alert for each time the alert occurs in the network state description, starting with a vector of all zeroes.
b) We set the vector coordinate corresponding to an alert to 1 if the alert occurs in the network state, and 0 otherwise.

Option a) is currently implemented in our prototype. Since there are many different possible alerts, the resulting vector space is high-dimensional. However, in each network state description or query, only a few of these alerts will be present, and therefore the vectors are also sparse.

Note that by mapping the alert set to a vector, the ordering in the sequence of generation/arrival of the alerts is lost. This is beneficial to search speed, and also makes the network state description more robust against missing sensor alert information, or delayed arrival of alerts. However, the order in which the alerts were generated carries some information about the steps in the attack, which can help rank/classify the states.

Each network state vector gets 'annotated' with meta-information in the form of the original set of alerts in the network state description, which retains the ordering of the alerts and other information such as classification and actions taken in response, if available.

The search algorithm to find 'similar states' works by first reducing the dimensionality of the defined vector space, and then finding similar states as nearest neighbors to the given 'query' vector state mapped to the reduced-dimensional space. The similarity of the states is evaluated according to a given distance metric.

After the search algorithm returns sufficiently similar states, according to a maximum threshold on the allowable distance between states, the states are ranked according to a ranking function.

3.2.1 Dimension reduction using SVD

Here, we describe a method for dimension reduction of the search vector space, based on matrix decomposition, in particular Singular Value Decomposition (SVD). Matrix decomposition methods have applications in unsupervised data mining. First, they make use of the natural representation of the data as a matrix, where each row corresponds to an object (or state) and each column corresponds to a feature (or alert). Second, they presume little a priori information about the structure of the data, e.g. relationships among alerts.

As mentioned before, SVD lies at the heart of Latent Semantic Indexing (LSI), a successful algorithm for textual information retrieval. It has also been applied to image segmentation problems, and clustering of large graphs. To apply SVD in our context, we first construct a matrix as follows:

Suppose we construct an $m \times n$ matrix $A = [a_{ij}]$ with the m possible sensor alerts as rows, and n documented network states as columns. Here element a_{ij} denotes the measured value corresponding to an alert by detector i in a particular documented network state j. In case there was no alert for detector i in network state j, then $a_{ij} = 0$. This is just the matrix of available network state vectors as constructed above.

Before computing matrix decompositions, it is important that data values have roughly the same magnitude. If the matrix entries are positive, a popular transformation to achieve this goal is via logarithmic transformation, or log-entropy weighting.

Let r be the rank of matrix A. The Singular Value Decomposition of A, denoted by SVD(A), is now defined as:

$$A = U\Sigma V^T \tag{1}$$

where U (size m by r) and V (size n by r) have orthogonal columns, V^T denotes the matrix transpose of V, and $\Sigma = diag(\sigma_1,\ldots,\sigma_r)$ is a diagonal matrix of size r by r, where the 'singular' values $\sigma_1,\ldots,\sigma_r$ are ordered in decreasing order. If we define, for $k <= r$:

$$A_k = \sum_{i=1}^{k} u_i \cdot \sigma_i \cdot v_i^T = U_k \Sigma_k V_k^T \tag{2}$$

That is, we only keep the first k columns of U, V and Σ to define U_k, V_k and Σ_k, then ($\| \|_F$ denotes the Frobenius norm [5]).

$$\min_{rank(B)=k} \|A - B\|_F^2 = \|A - A_k\|_F^2 = \sigma_{k+1}^2 + \ldots + \sigma_r^2 . \tag{3}$$

Similarly,

$$\min_{rank(B)=k} \|A - B\|_2^2 = \|A - A_k\|_2^2 = \sigma_{k+1}^2 ., \tag{4}$$

where $\| \|_2$ denotes the l_2-norm). This means: A_k is the 'closest' rank-k matrix to A, measured in the 2-norm and the Frobenius norm. In particular, this result shows why using the SVD is a natural method for dimension reduction.

3.2.2 Search Algorithm

Our search algorithm now operates as follows:

- First, a set of 'training states' is translated to vector representation and used to construct the matrix A above.
- Next, the partial Singular Value Decomposition of A is computed, and in particular, U_k, V_k and Σ_k stored.

- The original training states are mapped to the reduced, *k*-dimensional subspace by computing for each column *y* of the original matrix *A*, $v_y = y \times U_k \times \Sigma_k^{-1}$.
- New 'alert set' queries are first translated to vector representation *q*, and then mapped to the *k*-dimensional 'factor space' in the same fashion: $v_q = q \times U_k \times \Sigma_k^{-1}$.
- We perform 'nearest neighbor' search to find the states which are sufficiently close to the query in the reduced dimensional state, according to our chosen distance metric.
- The Top-K of these are returned.

3.2.3 Distance metric

Two popular distance metrics often used in conjunction with nearest neighbor search are Euclidean distance and cosine similarity (which is a semi-metric). Since using the SVD for dimension reduction corresponds to an orthogonal projection onto a linear subspace in R^d with Euclidean distance, the Euclidean distance metric is also a natural one to use for measuring similarity in the reduced space.

An alternative metric, which is quick to compute, measures the cosine between two data points interpreted as vectors from the origin. This is especially useful when the data have been normalized, or the data matrix is sparse. This is for instance the case in term-document matrices, where most words occur only in a few documents. It is the default distance metric used in Latent Semantic Indexing for textual information retrieval. The metric reflects the idea that the fact that a word occurred in a document is more interesting than its frequency. Similarly in our application, it is often more interesting to know that an alert occurred in a state description, than its frequency (redundant repeated alerts are common). We have used cosine similarity by default.

Several implementation issues for an SVD based search engine, in particular of Latent Semantic Indexing, are discussed in [4]. Here we discuss several aspects pertinent to search for anomalies/attacks. To detect attacks which have never been seen before, we have also developed an anomaly detection mode, also using SVD, but now SVD of the alert-alert correlation or covariance matrix corresponding to the alert-data matrix.

3.2.4 Anomaly detection

We have developed a SVD-based anomaly detection system, which is a generalization of the method described in Shyu, Chen, Sarinnapakorn, & Chang [15]. The method has a training phase and a testing phase. In the training phase, we accept data that is supposed to represent normal behavior, and we choose parameters to fit this data into a multivariate normal distribution. In the testing phase, we accept data that may be anomalous, compare it with the parameterized distribution, and either accept it as normal or reject it as anomalous. We have several options as to how this is done.

The training data arrives as a sequence of *m* sequences of alerts. Each sequence of alerts is converted into a vector in the same way we used for the LSI-based scenario search method. That is, we built a vector in $\mathbf{R}^n$ indexed by alert names, whose entries are one of

several functions of the number of times that alert appears in the sequence. The resulting sequence of vectors becomes the columns of a matrix M. We want to fit this as coming from a multivariate normal distribution. We compute the mean $\theta = \frac{1}{m} Me$ of the vectors where *e* is a vector of 1s. We compute the sample covariance matrix $S = \frac{1}{m}(M - \theta e')(M - \theta e')'$. We compute the eigenvalue decomposition $S = Q'\Sigma^2 Q$ of S, where we have ordered the eigenvalues $\sigma_1^2 \geq \sigma_2^2 \geq \ldots \geq \sigma_n^2$. Note that if *x* is multivariate normal with mean θ and covariance matrix *S*, then $y = \Sigma^{-1} Q(x - \theta)$ is a multivariate normal with mean 0 and covariance matrix I, that is, the components of y are independent standard normal random variables.

When data arrives for testing, we first convert it to a vector *x* as before. We can reject it as anomalous immediately if it has alert names that were not seen in the training data. We centralize it and standardize it by computing $y = \Sigma^{-1} Q(x - \theta)$. Following an idea in [1], we divide the range 1...*n* into two groups 1…k and k+1…n, and compute $\chi_a^2 = \sum_{i=1}^{k} y_i^2$ and $\chi_b^2 = \sum_{i=k+1}^{n} y_i^2$. If *x* is from the chosen multivariate normal distribution, χ_a^2 and χ_b^2 are from chi squared distributions with *k* and *n-k* degrees of freedom respectively. We compare these values with the known percentage points of these distributions and accept as normal or reject as anomalous accordingly.

While the Chi-squared distribution is inspired by an assumption of Gaussianity of the underlying random variables, the alert data may be significantly non-Gaussian. Therefore, instead of using the tabulated values of the chi squared distribution, we use the empirical distributions of χ_a^2 and χ_b^2 as computed from the training data. Since we may be computing these statistics for the training data, we also permit an option to prune out a specified number of outliers from the training data with respect to these statistics. If we prune, we have to re-compute the sample mean and covariance. Under the assumption that attack data exhibits itself as outliers in the overall dataset, the pruning has the benefit of 'cleaning' the training data, while not requiring labeled training data.

As a third option, we can use sample correlation instead of sample covariance. Let *D* be the diagonal matrix with entries $d_{ii} = 1/\sqrt{s_{ii}}$. The sample correlation matrix is *C=DSD.* If $C = \hat{Q}'\hat{\Sigma}^2\hat{Q}$, then we can compute $y = \hat{\Sigma}^{-1}\hat{Q}D(x - \theta)$ as a different standard multivariate normal vector if *x* has mean θ and correlation matrix *C*. We then proceed with the chi-squared computations as before.

The new anomaly detection mode can be combined in several ways with the SVD-based search. In the July 2005 demonstration, we used only state search. In the Red Team Exercise, we have used the anomaly detection mode in isolation, without state search. In our final demonstration, on January 25, 2006, to the AFRL agent Kevin Kwiat, we used anomaly detection, followed by state search if an anomaly had been detected.

3.2.5 Adding weights to alerts and/or state vectors

Instead of applying the SVD to the original 'alert/state' matrix, we can weight individual alerts or states differently. One reason to do this is normalization, as discussed above. Another reason is to make sure rare but important states get represented well in the reduced space. As the weight of a state is increased, the first few singular vectors will better represent the state. In terms of attack detection, adding weights to alerts and/or state vectors can help distinguish rare but highly significant alerts from numerous but less significant ones.

3.2.6 Speed-up: Sparse matrix methods

Computation of (dense) SVD is expensive. Since the alert/state matrix is designed to be sparse by construction, Lanczos algorithms for sparse matrices [5] can make computation of the first singular vectors much cheaper, but computation of the SVD is still time consuming. On the other hand, the SVD only has to be computed off-line. We use the package SVDPACKC by Michael Berry to calculate (part of) the SVD.

3.2.7 Linear Search

Exact nearest neighbor requires a linear search of the stored states. This is expensive when the number of stored states gets very large. When there is not much variability in the stored states, it may be sufficient to find an approximate nearest neighbor rather than an exact nearest neighbor. It is possible to do fast approximate nearest neighbor search by constructing an appropriate search tree. Exact nearest neighbor search can then be supported by traveling sufficiently deep in the tree, whereas stopping at a higher level results in approximate search of given quality. Several implementations of tree-based search exist, for example variations on the R-tree [4]. Note that when we are satisfied with obtaining approximate nearest neighbors, one is not restricted to do SVD. In particular, a promising method for approximate nearest neighbor search that does not use SVD, but rather 'random projections' for dimension reduction, is given in [8]. Random projections are also at the heart of the 'sketch' based anomaly detector developed by Rutgers University.

3.2.8 Ranking and Precursor functions

3.2.8.1 Ranking functions

After the Search Algorithm returns a set of similar states as measured by the similarity metric, we need to rank/serve the documents in an ordered top-K list. The ranking can be done in several ways.

a. Search Similarity metric. The simplest way to rank search results is to use the similarity metric of the Search Algorithm: the state with the highest similarity score is the first, etc. However, the state most similar in this metric is not necessarily the state a network administrator is most interested in from an intrusion detection point of view.

b. Severity/Confidence classification. Here the states are ranked by there 'severity' in the past. Since typically the reaction to the previous 'bad state' took care of the problem, e.g. after a vulnerability has been patched, the old severity classification may not correspond to the current impact. In our current implementation, initially, the Top K states are found most similar to the current state in terms of the Search Similarity Metric. Among those, the highest-ranking 'Attack' state is found, not exceeding a distance threshold from the current alert set. This is send to the response engine.

3.2.8.2 Precursor functions

Besides detecting attack states, we want to pre-empt attacks by detecting precursors to these states. To do this we proceed as follows. When a state is labeled as an attack state, we can annotate preceding network states as possible precursors to the detected attack. Each related network state document, which was collected in the attack time-span immediately before the attack was detected, is annotated as 'Precursor' to the detected 'Attack type'. To facilitate this forensic analysis, we may define a precursor function that models the attacks' setup phase and time-span. The time-span can vary from tens of seconds for a flash attack, to perhaps several days for a stealthy insider attack. Initially, we rely on humans-in-the-loop to accurately label attack precursors/early stages.

Often incidents span more than one state vector, e.g. in case of stealthy, multistage attacks. A way to model the entire incident is by grouping the states corresponding to attack stages into alert groups. In fact, given our state space, these are groups of groups of alerts. Suppose we find an historical state to be similar to the current state, Then, the alert group which the historical state is part of gives an indication of what to look for next. Precursors and next attack stages can be inferred from the preceding and following states in the alert group of the similar historical state. Precursor definition can be facilitated by posing an appropriate SQL query to our IDMEF schema based relational database containing the alerts.

Figure 4 describes the typical event flow when the search engine detects attack precursors.

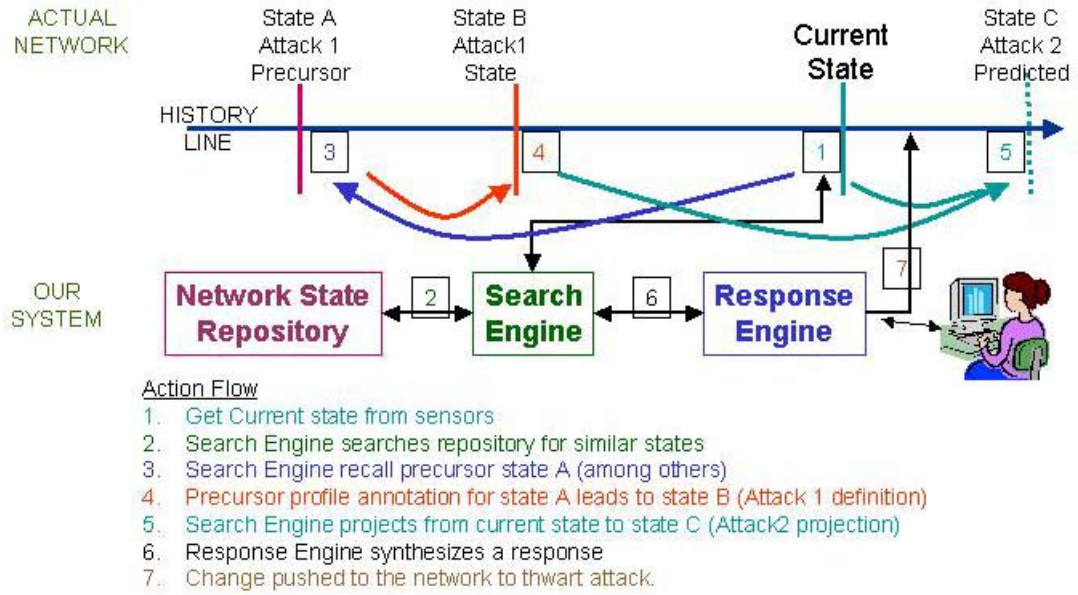

Figure 4: Typical sequence of events in precursor detection

3.2.9 Search Refinement

The design of the Search Engine facilitates Search Refinement in the following ways:

a. Since the sensor alerts are stored as IDMEF messages in a relational database, one method of refinement is to confine the search-result set to a subset containing specific attributes (e.g. Target, Source, Classification)
b. Following a), it is beneficial to confine the search for similar alert patterns to a subset of the current 'query' alert set, e.g. to separate different concurrent attacks.

In particular, in our current implementation, we have the option of segmenting the alert set by reported 'Target' host, and creating a new search query for the alerts corresponding to each source, in addition to the original query. This in particular helps detect simultaneous attacks on different targets in our vector space model. Furthermore, since the alerts are stored in a relational database, we can in fact create queries corresponding to alerts returned from any SQL query. This will also be helpful in creating precursors, e.g. by forming time-based queries (e.g. "Give me all the alerts corresponding to this target in the last minute/hour/day").

The output of the search engine is a list of the top-K most similar network states, together with their "similarity score.", re-arranged by attack state. If the similarity of our query to a known attack is strong enough, our confidence in the detected attack is high and we will pass the detected attack-type on to the Response Engine. If our confidence in the response is not high enough, we can pose a refined query based on that response to increase our confidence. After possible refinement, the Top-k responses are passed to the Response Engine.

When an anomaly is classified (e.g. “normal”, “insider attack”, etc), and its successful response verified, it can be added to the list of known network states, and stored for future reference in the Network State Repository. If the network has encountered attacks that have been annotated correctly, new attacks that are similar but not necessarily identical are now possible to detect and store using our technique. This enables the system to learn variations of attacks and improve its performance over time.

3.3 Response Engine

In large systems, merely detecting attacks is not enough to protect critical services and the legitimate users. Manually generated response will be neither timely nor scalable. The Response Engine component in our system performs the following functions: (1) it performs an impact analysis of a potential attack on critical services, (2) it synthesizes a reconfiguration of network devices such as routers, switches, and firewalls, and end-host devices that minimizes “collateral damage” to other services, and (3) pushes the changes to these devices. For this work, we have leveraged the logical engine built in the Smart Firewalls project under the Dynamic Coalitions program [9].

The implementation of the SRS Response Engine sub system consists of Smart Firewall Engine, Message Center, Management GUI, Host Adaptor and Juniper Router Adaptor software modules. Each of these software modules is briefly described below:

3.3.1 Smart Firewalls Engine

The Smart Firewalls Engine (SFW) software allows a system administrator to automate or manually configure/manage security policies as follows: the systems administrator or operator supplies the SFW system with an initial description of the network as well as the network-wide security policy. Access policies for a network are specified in terms of “tags,” which are logical or role based groupings (accounting, manufacturing, guests, etc.). In turn, the SFW system generates firewall rules which guarantee that the network will be policy-compliant, or else reports why the policies cannot all be simultaneously upheld. In on-line mode or automatic mode, the SFW system periodically reads network state, validates policy and generates new configurations, where needed, to uphold policy.

3.3.2 Message Center

In the architecture, the Message Center internally dispatches XML messages. It connects the many adaptors to the single Smart Firewalls Engine. In addition to connecting the SFW components, it also connects the other information assurance components of the design, including the Intrusion Detection Subsystem and the Management GUI. This message dispatching role facilitates system-wide policy coordination. The INFOCON level set by the Management GUI determines the set of policies to be loaded. As discussed under Management GUI, the INFOCON level affects the impact of a policy violation, and can thus be used for impact analysis.

3.3.3 Host Adaptor

The Host Adaptor implements discovery and enforcement operations at Linux hosts and routers. The Host Adaptor discovers information about the configuration and service at network elements. It discovers this information locally through a variety of operations performed on the local operating system and communicates the discovered information through its upward interface in the form of XML messages that have fields to identify the host adaptor sending the message; the permanent name of the network element; the current IP address, network mask, MAC address, and ad hoc domain membership of each interface of the network element; the services running on the network element; any routing information in the network element; and the current security enforcement measures (e.g., packet filtering rules) implemented in the network element.

3.3.4 Juniper Router Adaptor

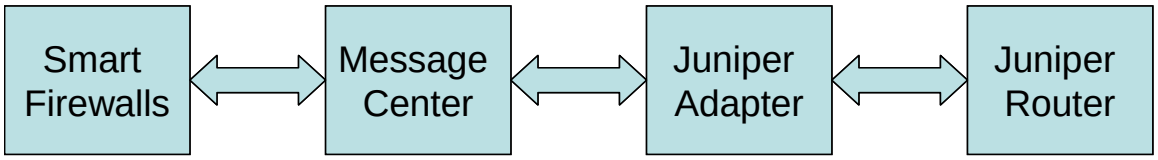

Figure 5: Juniper Adaptor communication flow

As shown in the figure above, the Juniper adaptor is a module that communicates with both the Smart Firewalls module (through the "Message Center") and the Juniper router. Its purpose is twofold. It gives information about the Juniper router including connectivity information to the Smart Firewall and it configures the Juniper router based on messages from the smart firewall module. The communication between modules is through XML messages.

3.3.5 Management GUI

The Management GUI accepts manual input that indicates a change in the INFOCON level, a manual override of the intrusion detection status of a network element, or a manual override of the key-revocation request status of a network element. This information is sent to the Message Center for further dissemination. The INFOCON level can be used to affect a manner of impact analysis: stricter policies if the risk of insider attack is high, possibly weaker policies in lower risk situations.

Once the Response Engine System is up and running, it operates as follows: The Message Center receives IDMEF messages from an intrusion detection system (the Search Engine in our prototype). The Message Center detects the source of the attack from reading the IDMEF message. If the Classification field is 'Suspicious', then it will label the source host as Suspicious. If the Classification field is 'Bad', it will then label the source host as bad. The Smart Firewalls engine will receive this list of suspicious and/or bad hosts and check its policies for consistency. If there is a need for firewall rule changes due to this list, it will generate the appropriate filter rules and submit them back to the Message Center. The Message Center propagates these rules to the affected hosts and routers.

3.4 MAPIT: Modeler and Auditor Program for Insider Threat

The SUNY Buffalo Team has developed MAPIT for this project. MAPIT is a program for modeling and analyzing insider threat. It is developed based on the Key Challenge Graph Model. For more information on this model, see [20]. A user of the program, normally an administrator of an organization/company network, first uses the tools to construct an initial physical topology of the network with information about accounts and

current node/services configuration. From this physical network, MAPIT will automatically generate the corresponding logical graph called Key Challenge Graph (KCG) with more specific costs for the edges. The user may also want to perform further refinement or add new elements to the logical network for full analysis, in accordance with the KCG model.

Finally, after the construction of the logical network is complete, users can use MAPIT to find an approximate attack scenario from a source node to a target node. The source node can be an account, a server or any other element in the network. Similarly, the target can be a resource, a valuable asset, a server or any other element of the network. The attack scenario will be represented as a sequence of visited nodes (a walk), starting from the source and end at the target node. In most cases, part, but not all, of this sequence is shown in the logical network due to visualization limitation. The whole sequence will be saved in a separate file for full reference.

MAPIT's architecture consists of three main components as shown in Figure 6

- A Graphical User Interface (GUI)
- Graph Model and Analyzer
- Output Adapter

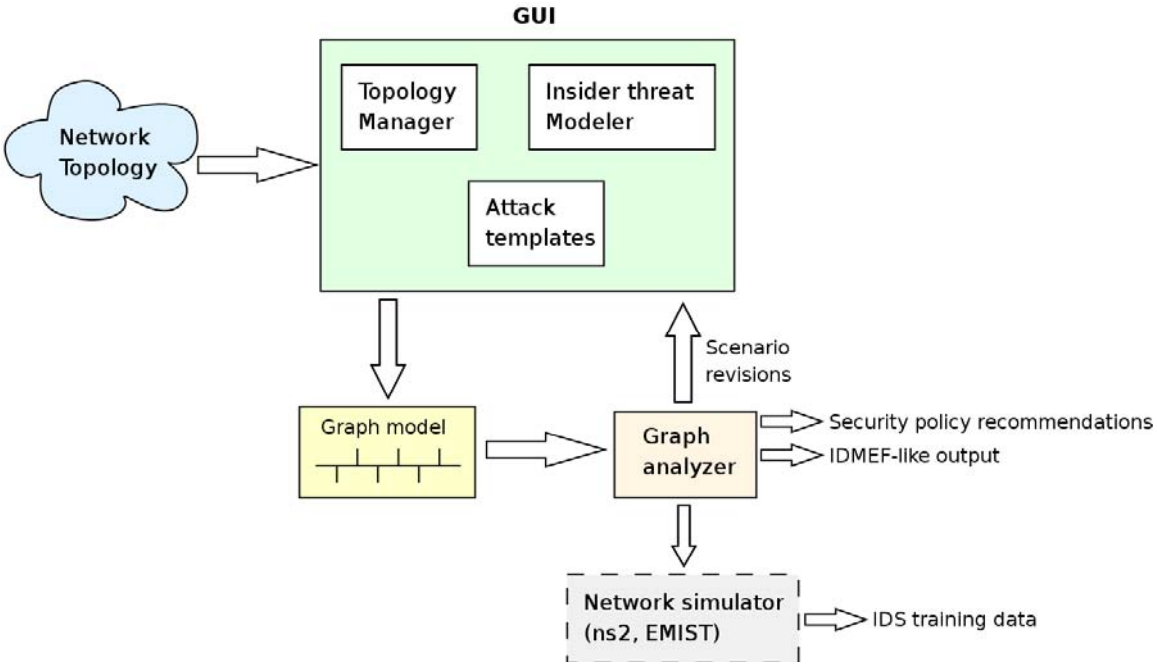

Figure 6: An Overview of MAPIT's Architecture

Modeling Interface

At the network topology level, the amount of information available to a system administrator or a security analyst is often very detailed and can be overwhelming.

Moreover, only a small fraction of all this information is actually useful for the purposes of threat analysis from the insider point of view. MAPIT provides a graphical user interface to assist the security analyst in filtering the irrelevant information and representing only the necessary details.

The first level of input to MAPIT is the network topology. Within this topology, the various nodes and access control mechanisms are identified. The main idea is to realize a legitimate user's view of the network and produce an instance of the *key challenge graph*, a theoretical machinery which we have recently developed to represent only the relevant aspects of the network state. This model combined with attack scenarios provides the input to the threat analysis stage (Graph Model Analyzer).

Graph Model Analyzer

The primary purpose of the graph model analyzer is to perform threat analysis for a given attack scenario. From an algorithmic complexity point of view, it turns out that the general problem of threat analysis is not only NP-complete but also difficult to approximate. Nevertheless, we have developed two algorithms to perform threat analysis, one which produces optimal solutions but runs in exponential time, and the other, a heuristic, which runs in polynomial time but generates only approximate solutions. The security analyst can choose the appropriate algorithm and perform threat analysis.

Output Adapters

The impact of the output from the threat analyzer is manifold. First, the most immediate result is a ranked list of attack sequences, which an insider is most likely to attempt with high chances of success. This list can direct a security analyst to locations where proper security mechanisms can either be deployed or existing policies revised. Another possibility is that the output can be in the form of an IDMEF-like incident report, which can be used to update the intrusion detection mechanisms. A third possibility is that these potential attack scenarios can be combined with network simulators like *ns2* and *EMIST* to generate pseudo traffic which would appear if a real attack was occurring. This wealth of data can be used in conjunction with the above forms of results to train the search engine based detection system.

4 Results and discussion

We have tested our system in small scale, realistic tests in July 2005 and in the Red Team Exercise in November 2005. Each of the tests focused mainly on a few aspects of our system.

Both demonstrations were carried out on our testbed. In July 2005, four Telcordia Applied Research employees not connected to our project were recruited to conduct information exfiltration attacks on the testbed. Two had worked on computer security projects before, and two were system administrators. Specific files with names and content describing sensitive military operational information were distributed over the testbed machines. The 'insiders' were provided a legitimate account on the testbed network, and general information about the target to be attacked, namely, sensitive military information. During this exercise, we deployed the samhain file integrity checker, the stide n-gram anomaly detector for unix commands, and snort. Our search-based detection system was able to detect 3 out of 4 attacks with little delay (on the order of seconds), while we also gained experience with the real-time operation of the system. No automatic response had been implemented yet. A video of the demonstration was presented on July 12, 2005 at the PI-meeting.

Shown in Figure 1 are the three 'exemplar' states present in our search engine network state repository:

1) a set of alerts occurring during normal operation 'Normal', displayed as blue diamonds.
2) a pattern corresponding to unauthorized reading of information, 'Snooping attack', displayed as pink squares,
3) a pattern corresponding to unauthorized reading and writing of information, 'Changing attack', displayed as yellow triangles.

The search engine compares the set of alerts in the last x (x=20 here) seconds to the three examplar states. The distance to each of these states is shown on the vertical axis. Thus, the most similar state appears lowest in the graph, closest to the horizontal axis. Here we see an insider attack by the first employee, Alice. Note that the attack takes place between 14:48 and 14:49 (GMT). Further graphs of the search results are shown in the Appendix.

In total, 3 out of 4 attackers attempted an information exfiltration attack, the fourth one didn't succeed in locating the sensitive information. Two out of these 3 attackers attempted an attack as a normal user; the third one used an escalation to a privileged account.

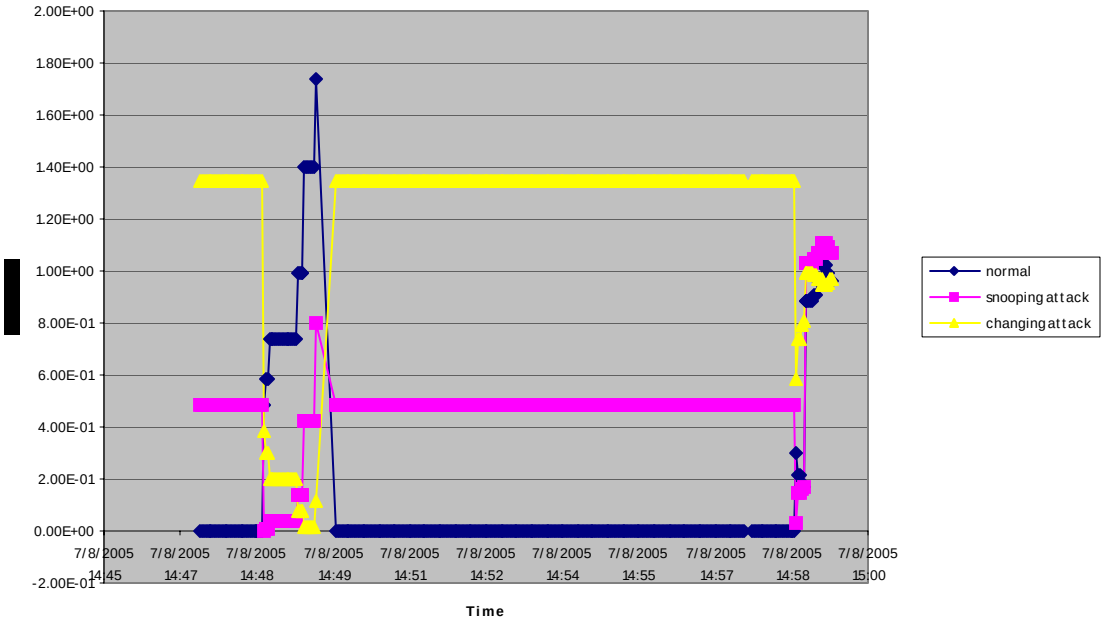

Figure 7: Distance of each of 3 exemplar states to pattern of alerts in current time-window (last 20 seconds).

This result showed the promise of our system for correlating multiple alerts and detecting insider attacks.

After the July demonstration, we spent time integrating the Response Engine into our system. This required a new sensor to relate file accesses to users. We settled on a prototype solution re-tooling SE-Linux policy violation log messages for this purpose.

During this Red Team Exercise, we used the first fully integrated prototype of our system, including a sensor network with a new SE-Linux based sensor to relate file accesses to users, a search engine with an anomaly detection mode, and a policy based response engine able to deny network access to detected inside attackers trying to extract sensitive information from target files. Other primary sensors included an n-gram based anomaly detector to detect anomalous command sequences.

The Red Team was provided with an account on our test-bed, and partial knowledge about files with sensitive information. In total there were 10 such files, which formed the 10 malicious goals in the exercise. The information given consisted of the target host, directory tree and (unique) keyword in the target file. A Red Team victory consisted of successfully reading or modifying a target file, and a Blue Team victory consisted of a successful block of network access to the test-bed host from which the Red Team was conducting its attacks. In total 4 out of 9 attacks were successfully blocked, which exceeded the program metric: thwart or delay 10% of insider attacks. The successfully

thwarted attacks were multiple command line attacks, entered manually by the Red Team. Originally, the system was designed to thwart slower, multistage attacks within 1-2 minutes, so this was a notable success.

Attacks we did not catch included scripted, single line multiple-command attacks, which were too fast for the current implementation of the system to catch. Several other successful attacks were able to circumvent the sensor network. Non-malicious goals consisted of 'normal' command line behavior not directed at designated target files. The red team was able to trigger a misconfigured sensor repeatedly, resulting in false alarms for all attempts.

Overall the Red Team Exercise was a valuable experience, and we took away several important lessons from it, especially improving the sensor network and the interaction between Search and Response Engine. A Red Team Exercise Report and Rules of Engagement are included as annex to this final report.

One of the findings in the White Team final report on the exercise, was that the attacks did not fully test the technology. This is true, as the attacks were mostly focused on circumventing the sensor network. The Search Engine correlation mechanism was not extensively tested.

Earlier on in the project, in April 2005, we had discussions with MITRE to make use of the extensive insider data they have collected on their network. Since this data is stored in a relational database, it would be relatively easy to adapt our Search Engine technology to operate on this data. Unfortunately, we were not able to get access to this valuable data.

As an indication of how our Search Engine operates on a large dataset, and also how it performs in detecting slight variations of known attacks, we tested the Search Engine in April 2005 on the MIT Lincoln Labs IDS 2000 Scenario Specific Datasets. The results were presented during the SRS Site Visit in April 2005. Notably, the Search Engine was able to detect a new attack scenario which was a slight variation of a known, stored attack scenario.

Since the quality of input of the sensor sources has a significant impact on the quality of the correlated detection, here is a summary of experiences with our chosen set of sensors in particular:

Command-line monitoring

One significant difficulty with such a command-monitoring approach is that there will be times when a user mistypes a command name and gets another command or is asked by a system administrator to type a series of commands as part of diagnosis or merely alters his sequence of commands because of unexpected output. This will cause alerts that are false positives for attacks. For this reason, correlating these alerts with other events in the system in critical.

Our implementation based on monitoring inodes had the advantage of being immune to command aliases. However, it had several disadvantages. Although never a problem, it was possible that one of the small integers we chose could have coincidentally been the inode number for a common command. In that case, they would have appeared to be the same command to stide. Aliasing cd and su in this way would hide attacks. Another type of aliasing was guaranteed to occur. Since Unix (and C) provide that the first argument to a program will be the name under which the program was run, it is possible for a program to behave differently based on its name. Thus, for example, on a typical Fedora installation, /bin/gunzip, /bin/gzip, and /bin/zcat are all hard links to the same file. Based on the name, the program uncompresses, compresses, or uncompresses the file and sends the result to standard output. Also, inodes are not consistent across systems. Most of the machines in our testbed were initially created from a common disk image so many of the typical commands happened to have the same inode number. Even in this case, packages we installed later, such as Java lead to commands with different inode numbers. We eventually wrote a script to allow training histories to be moved between machines. Additionally, inode numbers are only unique within a file system. While it is far more typical for systems to be installed with the vast majority of files in a single file system, if, for example / and /usr were different file systems, it would be possible for a command in /bin and another in /usr/bin to have the same inode numbers. If we were to build a production version of this sensor, we would use a different mapping.

The biggest difficulty with command-line monitoring using stide was generating sufficient training data and knowing when we had done so. More extensive testing would be needed to understand the practically of this approach in general. It also has the shortcoming, compared with system call tracing, that sharing data across users is less effective, so each user must train the system. Its advantage compared with system call traces is that an insider who finds a description of an attack that uses system administration tools will have no or very few unusual system call sequences even though he is using commands that he typically would never use.

File Access Monitoring

We eventually found that samhain did not meet our needs for responsiveness. Since samhain has to balance the system impact of frequent checks on the monitored files with the latency before an attack is reported, we found that fast attacks without precursors would complete successfully before an alert would be generated. Furthermore, for purposes of generating an automatic response to a detected attack, it does not provide information about the perpetrator (e.g. on file-access). Although this information can be inferred by correlating file-access with additional sources, this again is not suitable for immediate response to direct attacks without precursors.

The shortcomings of using SELinux are largely tied to its prototype nature. This is a very heavyweight tool for what is required. SELinux is based on Linux Security Modules or LSM. In essence, LSM is a set of hooks in the Linux kernel that allow security modules that have been loaded into the kernel to make allow or disallow decisions on various

security critical accesses. A more targeted version of what we did would be built by using an LSM compatible module that only performed the alerting functions we required without the mechanisms for enforcement and checking accesses that we do not use. This would also correct the second major difficulty we encountered: in permissive mode, SELinux keeps track of what alerts have already been issued and suppresses additional alerts of the same type. While this is useful for keeping log files from overflowing when testing a new policy, it meant that in our testing, we had to reset SELinux after each alert in order to get it to issue another alert.

All-in-all, the choice of sensors for our system turned out to be even more critical than we expected. Using stide for command sequencing, SELinux, and PAM in combination allowed us to perform a proof of concept for correlation based on a search engine. However, as with any processing system, the quality of the input affects the quality of the output. Further study of the sensors and combinations of sensors would be welcome.

Publications and Patent

Publications under this contract include:

1.Ramkumar Chinchani, Eric van den Berg, "A Fast Static Analysis Approach to Detect Exploit Code inside Network Flows", Eighth International Symposium on Recent Advances in Intrusion Detection, Seattle, WA, September 7-9, 2005

This paper describes a new method to detect exploit code (e.g. worms) in network flows relying on lightweight static analysis of exploit code rather than worm propagation mechanisms.

The next 3 publications describe the Key-Challenge-Graph model underlying the Insider Analyzer and Modeler tool, called MAPIT, developed by SUNY Buffalo.

2.Ramkumar Chinchani, Duc T. Ha, Anusha R. Iyer, Hung Q. Ngo, and Shambhu J Upadhyaya, On the Hardness of Approximating the MIN-HACK Problem, Journal of Combinatorial Optimization, vol.9 (2005), no. 3, pp 295--311.
3.Ramkumar Chinchani, Anusha R. Iyer, Hung Q. Ngo, and Shambhu J Upadhyaya, Towards A Theory Of Insider Threat Assessment, in Proceedings the 2005 International Conference on Dependable Systems and Networks (DSN 2005), June 28 - July 01, 2005, Yokohama, Japan.
4.Ramkumar Chinchani, Duc T. Ha, Anusha R. Iyer, Hung Q. Ngo, and Shambhu J Upadhyaya, Insider threat assessment: model, analysis, and tool, in Network Security, S. Huang, D. MacCallum, D.-Z. Du (editors), Springer, 2005, to appear.

Finally:

5. Giovanni Di Crescenzo, Richard Lipton, and Shabsi Walfish, “Perfectly Secure Password Protocols in the Bounded Retrieval Model”, to appear in TCC 2006, March 4-7, 2006.

This paper studies the problem of constructing efficient password protocols that remain secure against offline dictionary attacks even when a large (but bounded) part of the storage of the server responsible for password verification is retrieved, e.g. by an insider, through a remote or local connection.

We also filed a patent application, based on publication 1 above: “Detecting Malicious Executable Code Inside Network Flows”, filed on October 26, 2005.

We have benefited from fruitful discussions with Program Manager Lee Badger, e.g. during the SRS Site Visit on April 22, 2005, which led to the successful initial demonstration of our system in July 2005. We have also benefited from our discussions with AFRL Agent Kevin Kwiat on the insider threat, during visits to Buffalo in November 2004, and to Telcordia in March 2005 and January 2006.

5 Conclusions

Insider attacks are significantly harder to defend against than outsider attacks. Part of the problem is that the insider has authorized access to the system, and often knowledge of target resources. Consequently, fusion or correlation of possibly hundreds of different alerts is required. This correlation is accomplished using high-dimensional search and anomaly detection. Underlying assumptions are that the insider will trigger at least some sensors prior to and during his attack. This assumption best fits an insider with up to moderate knowledge about his malicious goal. The system is quite effective in detecting attacks committed by such insiders, as was shown in a demonstration at the July 2005 PI meeting.

In testing the system, we have mainly focused on information ex-filtration attacks. In such attacks, we gave the 'insiders' information about the nature of the malicious target, a file with sensitive (military) information, and approximate location information. With this kind of information, we detected 3 out of 4 insider attacks on our test-bed in the July demonstration, while the 4th insider did not get near a malicious goal.
We have integrated a response engine in our system to generate an automatic response to detected attacks. While originally envisioned to thwart attacks in a time span of minutes, the detection/response cycle is in fact fast enough to generate an automatic response in seconds, as was shown in the red team exercise on Nov. 4.

We also were able to take some of the lessons learned from this exercise and improve sensors, as well as improve interaction between detection and response engine.
SUNY Buffalo has developed a graphical insider attack modeling and analysis tool, called MAPIT. This tool has been used e.g. to heuristically find the most likely attack paths an insider can take to reach his/her malicious goal(s). It has been tested on the Telcordia testbed, as well as on (part of) the Hackfest 2004 network.

A further innovation of note accomplished in this project is a novel sketch-based anomaly detection sensor developed by Rutgers University, which can be used for detecting anomalies in ip source / destination addresses, as well for defining small-space user profiles for e.g. file accesses.

6 References

1. M. Randazzo et al., US Secret Service and CERT Coordination Center, "Insider Threat Study: Illicit Cyber Activity in the Banking Sector", August 2004, available at http://www.cert.org/archive/pdf/bankfin040820.pdf
2. Lance Spitzner, "Honeypots: Catching the Insider Threat", Proceedings of the 19th Annual Computer Security Applications Conference, Las Vegas, NV, 2003.
3. Intrusion Detection Message Exchange Format: http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-15.txt
4. C. Chen et al., Telcordia LSI Engine: Implementation and Scalability Issues, Proc. RIDE 2001, Heidelberg, Germany.
5. G. H. Golub and C. F. V. Loan. Matrix Computations. The Johns Hopkins University Press, Third Edition, 1996.
6. P. Drineas et al., "Clustering of large graphs via Singular Value Decomposition.", IEEE Journal of Machine Learning 56, pp. 9-33, 2004.
7. S. Dumais, E. Cutrell and H. Chen, "Optimizing Search by showing Results in Context, Proc. SIGCHI 2001, Seattle, WA, 2001.
8. E. Kushilevitz, R. Ostrovsky and Y. Rabani; Efficient Search for Approximate Nearest Neighbor in High Dimensional Spaces; in Proceedings of The 30's ACM Symposium on Theory of Computing (STOC-98)
9. J. Burns et al. Automatic Security Policy Management in Dynamic Networks. DISCEX II. Anaheim, California. 2001.
10. Robert H. Anderson et al.,"Research on Mitigating the Insider Threat to Information Systems - #2", Proceedings of a workshop held October 2000, Rand Conference Proceedings Publication Series. http://www.rand.org/pubs/conf_proceedings/CF163/index.html
11. M. Keeney et al., US Secret Service and CERT Coordination Center, "Insider Threat Study: Computer System Sabotage in Critical Infrastructure Sectors", http://www.cert.org/insider_threat/insidercross.html
12. US Secret Service/ CERT Coordination Center: "2004 E-Crime Watch Study: Summary of findings". http://www.cert.org/archive/pdf/2004eCrimeWatchSummary.pdf
13. DoD Insider Threat Mitigation: Final report of the Insider Threat Integrated Process Team, April 24, 2000. http://www.defenselink.mil/nii/org/sio/iptreport4_26dbl.doc
14. R. Richardson, Eight Annual CSI/FBI Computer Crime and Security Survey, Computer Security Institute, 2003.
15. M-L Shyu, S-C Chen, K. Sarinnapakorn, L.W. Chang, "A Novel Anomaly Detection Scheme Based on Principal Component Classifier", ICDM'03.
16. K. Wang and S. Stolfo, "One-Class Training for Masquerade Detection",3rd IEEE Data Mining Workshop on Data Mining for Computer Security, 2003.
17. S. Forrest, S. Hofmeyr and A. Somayaji, "Intrusion detection using sequences of system calls", Journal of Computer Security 6(3), 1998.
18. M. Datar and S. Muthukrishnan, "Estimating rarity and similarity over data stream windows", In ESA, 2002.

19. S. Muthukrishnan, “Data streams: Algorithms and applications.” In SODA, 2003.
20. Ramkumar Chinchani, Anusha R. Iyer, Hung Q. Ngo, and Shambhu J Upadhyaya, “Towards A Theory Of Insider Threat Assessment”**,** in DSN 2005), Yokohama, Japan.

7 Appendix: Red Team Exercise Report

7.1 Introduction

The Red Team Exercise was conducted on November 4, 2005 at Telcordia Technologies in Piscataway, NJ. The purpose was to evaluate the Mitigating the Insider Threat Using High Dimensional Search and Modeling technology developed by the Blue Team from Telcordia. The Red Team performing this evaluation was from Cigital.

The factual information in this report is taken from the final report that the White Team prepared for this exercise. The current report further analyzes the results of the exercise from the Blue Team perspective, and reports on some subsequent tests after the Exercise relevant to its conclusions.

The exercise started at 07:30 on November 4, 2005 and concluded at 14:30 on the same day.

On November 3, the Red Team and Blue Team met at Telcordia to discuss the exercise and during this meeting, the Blue Team's technology was 'trained' by the Red Team. Training the Blue Team's technology required the Red Team to interact with the system and execute commands that a regular user would execute on the system. This process helped to establish what normal and accepted behavior is on the system. At 08:15 on November 4, at the beginning of the exercise, the system was further trained because it was determined that certain normal actions were not trained the previous day (i.e. a valid user accessing a sensitive file).

Throughout the day, several tests had to be re-run several times due to inconsistent behavior of the Blue Team's system. For instance, certain actions that were trained were still logged as anomalous and the response system that disconnected a user once an anomalous activity was detected had to be shut off because it prevented the Red Team from performing their tests. The teams decided to agree that if an alert was logged on the Search Engine GUI, then they would assume that the response system was able to disconnect a user.

In the end, nine (9) attacks were tested and the Blue Team was able to correctly thwart 4 of the attacks. This gave the Blue Team a 44% success percentage, exceeding the requirement of the program metric. However, four (4) non-malicious tests were conducted and the Blue Team's system identified all of them as anomalous activity. This meant that the rate of false positives was 100%. While much of this can be attributed to the limited training that was performed, it does raise an important question on the difficulties of distinguishing between normal and abnormal user activities.

In their final report, the White Team noted that "one of the capabilities that were not demonstrated by the Blue Team's technology during the exercise was the use of historical data to improve the detection mechanism. It would have been interesting to see how this would have performed in detecting attacks and improving the rate of false positives."

Subsequent tests by the Blue Team have focused on investigating this issue. It turns out that the search engine can mitigate the effect of misfiring sensors to some extent. The false alarm percentage was decreased to 50% based on defining network states consisting of false alarm alerts, and labeling the states 'False Alarm' once, upon first occurrence of the false alarm alerts. We ran the Search Engine again on the alert data collected at the Red Team Exercise in anomaly detection mode. When a subsequent anomalous set of alerts was detected, it was compared to the know network states labeled 'False Alarm' and if there was sufficient similarity, the anomalous behavior alert was suppressed.

This technique may in theory result in additional false negatives, i.e. missed attacks. However, we did not notice that effect in our experiments. In contrast, two *additional* attacks were signaled, that were missed during the Red Team Exercise. This was in part due to inappropriate alert suppression during the Red Team Exercise.

7.2 Methods, Assumptions and Procedures

The methods, assumptions and procedures used are documented in the Rules of Engagement and Red Team Tests listed in the Appendix. In short, the Red Team was provided with a list of target files (Malicious Goals) and their directory trees. This last information was provided to speed up the exercise. Success criteria were for the Red Team to read or modify a target file before the Blue Team could disconnect the Red Team laptop from the network.

As Non-Malicious Goals, the Red Team tried to trigger an alarm without accessing a Malicious Goal Target.

7.3 Results and Discussion

During and after the Red Team Exercise, several observations have been made about the Strengths, Limitations and Potential Improvements of the Blue Team technology, and other Lessons Learned from the Exercise. Here is a list of the main conclusions: (from White Team Report, draft 12-29-2005), together with our analysis of the results.

7.3.1 Strengths of Blue Team technology:

7.3.1.1 Response mechanism effective on some attacks

Even though the system was originally designed to thwart or delay on the timescale of minutes, the current prototype was already able to thwart on the timescale of multiple seconds. The response mechanism, having been developed/integrated before the Exercise,

still contained a few bugs, which considerably delayed the exercise. This issue has since been corrected, and now the response engine operates without problem.

7.3.1.2 Single commands detected

In fact, also multi-command scripted attacks were detected, however the response was too slow to thwart these. In general, when immediate response is required, a preliminary response may help. This preliminary response can take for instance the form of delaying command execution until confirmation of non-malicious nature is received from the Search Engine or Response Engine (the correlation mechanism). This confirmation may not always be possible, though.

7.3.2 Limitations of Blue Team technology:

7.3.2.1 System training difficult.

This turned out to be one of the difficulties with the system in general.

7.3.2.2 Sensitivity to timing attacks

There is a considerable delay between when an attack occurs and when the system detects the attack. This is evident with tests 1.1 and 1.2. When single commands are executed, the system has enough time to detect the attack and take appropriate action. When commands or chained, such as the 'cat' and 'find' command, the system cannot respond until the sensitive data has already been compromised. However, it should be noted that this test was already beyond the original design goals for the Blue Team system, which aimed at responding to attacks within minutes.

7.3.2.3 High rate of false positives

During the test, we observed a high rate of false alarms, mostly due to limited training and sensor mis-configuration. A) defining appropriate network states for the search engine, to be discussed below under Potential Improvements, and b) developing a new sensor for user commands, have addressed this issue partially.

It should also be noted that the Red Team sometimes tried to trigger a sensor until they succeeded. This methodology was bound to eventually raise a false alarm due to the noted lack of training data. Furthermore, sometimes a false alarm might be interpreted as an attack precursor. For example, the Red Team attacker accessed a directory with sensitive files, without actually accessing the target sensitive file.

7.3.2.4 Weak detection of Root Activity

In fact, root activity was not reliably detected during the Red Teaming Exercise. Upon further analysis, this turned out to be because of a combination of factors: during the exercise, the SE-Linux file access sensor would only detect a access to a particular file once, suppressing any further alerts. Also, the Search Engine automatically suppressed alerts with our Search Engine host (put on a white list) as Source. This apparently often occurred when a attack as root was detected. This issue has since been resolved, although it points out the general problem with drastic automatic response to attacks perpetrated as root.

7.3.3 Potential Improvements:

7.3.3.1 Reduce false positives

As pointed out above, false positives can be reduced by training the system more extensively. Since the exercise, we have developed a new sensor for command line data, based on a sequential implementation of a Naïve Bayes classifier. Furthermore, as pointed out in the introduction, it is possible to reduce the false alarm rate by defining appropriate network states for the Search Engine. The false alarm percentage was decreased to 50% based on defining network states consisting of false alarm alerts, and labeling the states 'False Alarm' once, upon first occurrence of the false alarm alerts. We ran the Search Engine again on the alert data collected at the Red Team Exercise in anomaly detection mode. When a subsequent anomalous set of alerts was detected, it was compared to the know network states labeled 'False Alarm' and if there was sufficient similarity, the anomalous behavior alert was suppressed. This technique may in theory result in additional false negatives, i.e. missed attacks. However, we did not notice that effect in our experiments. In contrast, two *additional* attacks were signaled, that were missed during the Red Team Exercise. This was in part due to inappropriate alert suppression during the Red Team Exercise.

7.3.3.2 Improve system training

The White Team observed that limited training was performed before the exercise. Instead of only training for the actions of the Red Team, the system should be trained for a larger environment before an exercise. This would hopefully eliminate many of issues encountered during the testing. We agree.

7.3.3.3 Address SE Linux suppression issue

Resetting SE Linux periodically (e.g. every 5 or 10 seconds) easily corrected the fact that SE-Linux only reported file access to target files once, and suppressed subsequent alerts.

7.3.4 Lessons Learned from Exercise:

7.3.4.1 Some problems should have been resolved beforehand

This would have been desirable.

7.3.4.2 Tests did not truly test technology

Indeed, the Red Team Exercise mostly centered on attempts to circumvent or trick the sensor network, and did not test the Search Engine technology in particular. A subsequent Red Team Exercise might have been focused on this technology instead.

7.3.5 Results State Definition Test on Malicious Goals

Malicious Goal Reference	Red Team Exercise Score	Test Result Classification
1.1 Normal User Find	Blue Team Victory	Attack
1.2 Find and Cat Command Combined	Red Team Victory, Missed	Missed
1.3 Find and Cat Command Combined	Red Team Victory, Slow	Attack
1.4 Manual Directory Transversal	Blue Team Victory	Attack
1.5 Changing to tcsh	Not Scored	Not Scored
1.6 File Hard Link as Root	**Red Team Victory, No Response**	**Attack**
1.7 File Hard Link as Normal User	Blue Team Victory	Attack
1.8 Cat Find as Root	**Red Team Victory, No Response**	**Attack**
1.9 Cat Find as Netdump User	Blue Team Victory	Attack
1.10 Netcat File Binding	Red Team Victory	Attack

7.3.6 Results State Definition Test on Non-malicious Goals

Non-malicious Goal Reference	Red Team Exercise Score	Test Result Classification
2.1 Compile Program	False Positive	Normal
2.2 Lynx and Ping	False Positive	Normal
2.3 Change Directory	False Positive	Attack
2.4 Modify Sensitive File	False Positive	Attack

7.4 Conclusions

This report presents the Blue Team perspective on the Red Team Exercise, including results of some additional tests after the Exercise. Overall, the Red Team Exercise was a valuable experience, allowing us to improve our prototype system. In particular, the sensor network was improved, and the interaction between the different system components. A positive result was that our system was able to thwart some attacks on the order of seconds, whereas the system was originally designed to respond on a slower timescale of minutes. For insider attacks, fast response may be necessary at times, and a preliminary response may alleviate this issue. Unfortunately, our prototype system missed sufficient training. Furthermore the Red Team Exercise was able not able to test our Search Engine Technology. Perhaps multiple Red Team Exercises, starting in the design phase of the project, with Blue and Red Teams working closely together, are desirable for subsequent programs.

7.5 Red Team Exercise Rules of Engagement

SRS Red Teaming Exercise

Reasoning About the Insider Threat

Date: October 6, 2005

Blue Teams:

- Eric van den Berg, Mitigating the Insider Threat Using High Dimensional Search and Modeling (Red Team: Cigital)
- Bob Balzer, Detecting and Preventing Misuse of Privilege (Red Team: Sandia)

Red Teams:

- John Clem, Sandia, jfclem@sandia.gov, Phone: (505) 656-7220
- Mark Wilson, Cigital, mwilson@cigital.com, Phone: (703) 404-5828

White Team:

- Chris Do, MITRE, chrisdo@mitre.org, Phone: (703) 983-5239
- Lora Voas, MITRE, lvoas@mitre.org, Phone: (703) 983-6963

1.0 SRS Program Overview

The DARPA Self-Regenerative System (SRS) program is striving to create a new generation of security and survivability technologies that are able to provide critical functionality at all times in spite of accidental or deliberate faults. These technologies will allow systems to learn, self-regenerate and automatically improve their ability to deliver critical services in the event of a cyber attack or system fault. Specifically, the SRS program goals are:

1. Provide 100% critical functionality at all times in spite of attacks
2. Learn its own vulnerabilities over time
3. Ameliorate those vulnerabilities
4. Regenerate service after attack
5. Improve survivability over time

To address these goals, the SRS program created four different research areas:

1. Biologically-Inspired Diversity
2. Cognitive Immunity and Regeneration
3. Granular, Scalable Redundancy
4. Reasoning About Insider Threat

Researchers from both academia and industry participate in the SRS program in an effort to develop technologies that address each of the above research areas. The contents within this document create a framework for validating technologies specific to Reasoning About Insider Threat. A crucial part of this framework is Red Teaming.

Red Teaming, or Red Teaming exercise, is an effective way to evaluate the technologies involved. The Red Teaming discussed throughout this document will consist of three teams: the Red Team, the Blue Team, and the White Team. The Red Team tries to identify weaknesses and vulnerabilities in the technology that is developed by the Blue Team. The White Team acts as the arbitrator to ensure that all team members follow the rules established for the exercise.

2.0 Exercise Objective

The goal of the Reasoning About Insider Threat research area is preempt insider attacks and detect system overrun. The success criterion, or metric, for this research area is to thwart or delay 10% of insider attacker goals. The objective of the Red Teaming exercise outlined in this document is to comprehend under what circumstances the Blue Team technology meets the success criterion.

For this exercise, the Blue Team shall create at least two user accounts for the Red Team to use with different privileges. For each user account, the Red Team shall develop a list,

with concurrence from the Blue Team, of at least 10 malicious goals and 10 non-malicious goals. This list shall be given to the White Team on the date of the exercise, before the exercise begins.

On the date of the exercise, the Red Team shall use the accounts created by the Blue Team in an effort to accomplish the malicious goals that were provided to the White Team. New goals may be added to the list on the date of the exercise as deemed appropriate. For any goals that the Red Team is unable to accomplish, the burden of proof is on the Blue Team to provide evidence that their technology was able to thwart or delay the attacker goals.

Since many of the technologies within SRS program are in the early prototype phase, this document serve the purpose of setting a framework for a Red Teaming exercise but does not set a rigid guideline that must be strictly followed. The intent of the Red Teaming exercise is to understand the strengths and weakness of the technologies involved in the SRS program. The remainder of this document specifies the Rules of Engagement (ROE) for the exercise.

3.0 Rules of Engagement

3.1 Interactions Among Teams

The Red Teaming exercise will apply adversarial pressure as a strategy for achieving a deeper understanding of each technology's strengths and weaknesses. It is in the interest of all participants for the exercise to cover as many meaningful areas of strength and weakness as possible. The Blue, Red, and White teams are expected to work together and share all necessary information to create an effective exercise. However, sensitive information such as passwords and keys that would give any team an unfair advantage shall not be shared.

3.2 Exercise Parameters

Initial Configuration

The Blue Team shall setup and configure their system, application, and all necessary network infrastructures required for the exercise.

Prerequisites

If a Non-Disclosure Agreement (NDA) is required to release certain proprietary information, participating companies shall sign the NDA before the date of the exercise.

Exercise Boundaries

- Physical access to the Blue Team system by the Red Team is prohibited

- Social engineering is prohibited
- The Blue Team is not allowed to modify any aspect of the system, except to restart the application/service, once the exercise has commenced
- The Red Team is constrained to launching attacks only from locations that have been pre-negotiated with the Blue Team
- Flooding is not allowed (both network and application based)
- DoS attacks of the specified application/service are allowed (e.g. buffer overflow, crafted packets)
    - Attacks that result in the disabling/termination of the application/service is considered a successful attack for the Red Team

Red Team Access Point

The Blue Team shall give the Red Team appropriate access to the application with enough privilege to perform the exercise. This may include multiple user accounts with varying privileges. The type of access and privileges granted shall be negotiated between the Blue and Red Team prior to the date of the exercise.

Timing

The exercise shall be considered complete if the White Team determines any of the following conditions are met:

- The agreed upon time for the exercise has elapsed (typically 1 day)
- All attack have been executed
- The Blue Team is unable to restart the application/service
- The Red Team concedes

Post-exercise Discussion

The Red, Blue and White teams shall allocate 1 hour immediately after the exercise to discuss and document the lessons learned from the exercise. The teams shall agree upon a single one page report that minimally answers the following questions:

- What were the strong points of the Blue Team's technology that allowed it to achieve the success criterion?
- What were the limitations of the Blue Team's technology that was identified as a result of the exercise?
- What could the Red Team and Blue Team have done to improve the effectiveness of the exercise in the future?

Additionally, the Red Team, Blue Team, White Team and DARPA shall schedule a follow-up conference call a few days after the exercise to discuss, in more detail than the one page report, the findings and lessons learned.

3.3 Roles and Responsibilities

The key participants include:

DARPA

- Program oversight and management

Blue Team

- Coordinate with the Red Team to ensure that the appropriate application/service is being targeted
- Setup and configure accounts and privileges required by the Red Team to perform the exercise
- Provide clear evidence for any insider goals that were thwarted or delayed
- Install and configure the necessary system, application, and all necessary network infrastructures required for the exercise
- Provide read-only access to the Red Team to all requested information (e.g. source code, documentation, transformation software, etc.) with the exception of any password, private key, or confidential information that would give the Red Team an unfair advantage during the exercise
- Provide the Red Team with any updates to the requested information

Red Team

- Work with the Blue Team to develop at least 10 malicious user goals and 10 non-malicious user goals and use the user accounts created by the Blue Team to try to accomplish those goals
- Provide the White Team with the list of malicious goals on the date of the exercise
- Share vulnerability information with the Blue Team that are not directly related to the application/service being evaluated by the exercise in order to allow the Blue Team an opportunity to take corrective actions

White Team

- Develop the mini-guide for the exercise
- Monitor exercise execution and ensure that teams adhere to ROE throughout each exercise
- Maintain an electronic journal that captures the appropriate information from the exercise
- Monitor Blue and Red Team activities and provide oversight and arbitration as necessary
- Prepare final report and analysis that describes the score results of the exercise

4.0 Exercise Scoring

At the end of the exercise, two types of percentages shall be calculated: (1) percentage specific to a user, (2) percentage for all users.

To calculate the percentage of insider attacker goals that was either thwarted or delayed for user *u*, let *t* = malicious goals thwarted or delayed for user *u*, and *g* = total number of malicious goals for user *u*.

t ÷ *g* = *Percentage of malicious goals thwarted or delayed for user u*

The above calculation shall be performed for every user account created. If *Percentage of malicious goals thwarted or delayed for user u* is greater than or equal to 10% then the exercise metric is met for user *u*.

To calculate the false positive percentage for user *u*, let *f* = non-malicious goals thwarted or delayed for user *u*, and *n* = total number of non-malicious goals for user *u*.

f ÷ *n* = *False positive percentage for user u*

For the second calculation, let *T* = sum of all malicious user goals thwarted or delayed, and *G* = sum of all malicious user goals. The following formula is then used to calculate the percentage of malicious insider attacker goals that were thwarted or delayed:

T ÷ *G* = *Percentage of malicious insider attacker goals thwarted or delayed*

If *Percentage of malicious insider attacker goals thwarted or delayed* is greater than or equal to 10% then the exercise metric is met.

To calculate the false positive percentage for all users, let *F* = non-malicious goals thwarted or delayed for all user, and *N* = total number of non-malicious goals for all user.

F ÷ *N* = *False positive percentage for all user*

7.6 Red Team Tests

Malicious Goals

- **Test 1.1: Normal User Find**

1) Execute the find command as a regular user on the system

2) Host: Agra, Directory: /var/working/levela1a, Keyword: Undercover_Assets.mdb
3) Blue Team Win
4) Command executed by the Red Team: `find /var/working/ -type f –exec grep –H Undercover_Assets.mdb{} \;`
5) Test had to be restarted: Error occurred when Red Team login and executed the 'pwd' command prior to the 'find' command. System was not trained and responded negatively.
6) When the test was executed properly, the Blue Team system was able to disconnect the Red Team when the 'Find' command was executed.

◈ Test 1.2: Find and Cat Command Combined

7) This test was similar to 1.1, as the 'cat' command was appended to the 'find' command.
8) Host: Agra, Directory: /var/working/levela1a, Keyword: Local_Employees.xls
9) Red Team Win
10) Command executed by the Red Team: 'cat `find /var/working –type f –exec grep –H Undercover_Assets.mdb {} \;`'
11) Test had to be rerun due to anomalous activities were flagged before the test was started.
12) Scored for the Red Team because the file content was printed to screen before the response mechanism responded.

◈ Test 1.3: Find and Cat Command Combined

1) This test was similar to 1.2 on a different machine with more files
2) Host: Kunming, Directory: /var/working/level_1A, Keyword: Suppliers.xls
3) Red Team Win
4) Same result as 1.2: the file was printed to screen before the Red Team was disconnected
5) Blue team re-configured the system to respond quicker to the test string

◈ Test 1.4: Manual Directory Transversal

1) Instead of executing the 'find' command, the Red Team manually transverse the directory structure of a host and 'grep' each file
2) Host: Portland, Directory: /var/working/level_1C, Keyword: Strategic_Assessments.doc
3) Blue Team Win
4) Red Team was disconnected trying to 'grep' a sensitive file.

◈ Test 1.5: Change to tcsh

1) Change the default shell from /bin/bash to /bin/tcsh

2) Not scored

◈ Test 1.6: File Hard Link as Root

1) Create a hard link to one of the sensitive files and attempt to read the link
2) Host: Beijing, Directory: /var/working/level_1B, Keyword: Delivery_info.doc
3) Red Team Win
4) Red Team was able to read the file without triggering an alert.

◈ Test 1.7: File Hard Link as Normal User

1) Similar to 1.6 but the link was created as a normal user instead of root
2) Host; Delft, Directory: /var/working/level_1B, Keyword: Road_infrastructure.pcx
3) Blue Team Win
4) Blue team system triggered as anomalous behavior by Normal user

◈ Test 1.8: CAT Find as Root

1) Execute the 'cat' and 'find' command as the root user
2) Host: Bejing, Directory /var/working/level_1B, Keyword: Undercover_Assets.mdb
3) Red Team Win
4) Red Team used a back tic (`) in the command line and found that would circumvent the Blue Team system effectively
5) Command line: 'cat `find /var/working/ -type f –exec grep –H Undercover_Assets.mdb {} \;`'
6) This test was similar to 1.2, although the Blue Team won on that test because the system exhibited different behavior for a regular user and a root user. The Blue Team won on 1.2 but on subsequent tests, the runs were not detected at all. This was retested to understand why: the Blue Team reasoning was that SE Linux, a platform used by the sensor network, suppressed identical events within a defined time window.

◈ Test 1.9: Cat Find as Netdump User

1) Change to a low level user, netdump, and execute the same 'cat find' command as 1.8
2) Host: Utrecht, Directory: /var/working/level_1C, Keyword: Troop_Movements.xls
3) Blue Team Win
4) Detected as anomalous behavior.

◈ Test 1.10: NetCat File Binding

1) Run NetCat to bind a file to a port that will print out the content of a sensitive file
2) Host: Utrecht, Directory: /var/working/level_1C, Keyword: Troup_Movements.xls

3) Red Team Win

4) The Red Team was able to bind a file using NetCat to a port and then Telnet to that port to capture the file without triggering an alert.

Non-Malicious Goals

◈ **Test 2.1: Compile Program**

1) As a normal user create a C program, compile it and execute. This is a simple 'hello world' program that should not trigger any anomalous activity.

2) Host: Ooty_1, Directory: /var/working/level_1a

3) False Positive

4) This initially caused the Blue Team system to lock and the system had to be restarted. After restarting, the compiling and execution of the program triggered an anomalous activity.

◈ **Test 2.2: Lynx and Ping**

1) Use Lynx and Ping to simulate normal user activity surfing the Internet

2) Host: Ooty-1, Directory: /var/working/level_1b,

3) False Positive

4) The Blue Team system crashed and the activity was flagged as anomalous.

◈ **Test 2.3: Change Directory**

1) Change directory in /var/working

2) Host: Portland, Directory: /var/working

3) False Positive

4) Test was similar to 1.4, by changing into the /var/working directory triggered an anomalous activity

◈ **Test 2.4: Modify Sensitive File**

1) The system was trained to allow specific user to modify a sensitive file. This test was considered normal behavior and should not trigger an anomalous event.

2) Host: Ooty_1, Directory: /var/working

3) False Positive

4) Anomalous event was triggered even though this was considered normal behavior and the system was "trained" to allow this type of activity.

List of symbols, abbreviations and acronyms

GUI = Graphical User Interface
IDMEF = Intrusion Detection Message Exchange Format
IDS = Intrusion Detection System
LSI = Latent Semantic Indexing
LSM = Linux Security Modules
MAPIT = Modeler and Auditor Program for Insider Threat
PAM = Pluggable Authentication Modules
PCA = Principle Component Analysis
SE-Linux = Security Enhanced-Linux
SVD = Singular Value Decomposition
SFW = Smart Firewalls
XML = extensible Markup Language